APPLIED HOMEWORK #1

Design Using FPGAs

1 Description and Homework Objectives

The report on this work is to be submitted by the AHW#1 report deadline (listed on the course webpage). You should review any other Quartus-related materials linked there as well. Please note that, although this applied homework is significantly shorter than the remaining four, it still will require some time to complete. You will demonstrate the work you complete for your report in the "lab" section for which you registered. This will take place in the week where "Applied HW#1 DEMO" is listed on the course schedule (given on the course webpage). The work for this report should be completed **individually**. At the demo, you can form groups of two (from the same lab section) for the demo work and remaining Applied Homeworks.

After this homework, you should be able to use the Quartus tool to:

1. Input designs as a schematic in the Quartus software
2. Perform functional and timing simulations of your design
3. Compile your design into a bit-file to program an FPGA
4. Use the programmer to program the FPGA and verify operation of your design on the FPGA

2 Background: FPGAs and Quartus

A designer can choose from many different implementation technologies for a given application. The most basic choice is between using a microprocessor and a custom hardware circuit. In programming courses, your instructor essentially decided in advance that the implementation of your project was best done using a microprocessor running your software. In this course, we teach the fundamentals of digital design, and implement applications in *hardware*.

An engineer can implement hardware in several ways: full custom, semi-custom (standard cell), and field-programmable. In a full custom design, the engineer designs the entire circuit from the transistor level using no pre-designed elements. This method has the most potential performance but is the most expensive both in time to design and initial cost for one chip. In a semi-custom design, the designer uses a library of standard cells (e.g. logic gates, memory, etc) combined with automated tools to create the circuit. This method produces designs much more quickly than full custom, but is still expensive for initial cost (still need the factory for fabrication). The final method is to use field-programmable chips. In this method, the designer uses a pre-made chip able to implement many different circuits, such as a field-programmable gate array (FPGA). The designer creates their circuit and uses automated tools to map the circuit to the FPGA. This method is the quickest and has the smallest initial cost; however, it also has the worst potential for performance. In this class, because we want to implement many different small circuits in a short period of time, we will use FPGAs.

2.1 Altera Quartus Tools

In this homework, we will use the Altera Quartus Prime 16.0 tools for performing design work. If you would like to run these tools on your PC at home, you can download the Quartus Prime 16.0 Web Edition from Altera's website.

Note: Even if you use Quartus on your own machine, be sure that you get your CAE account several days before your demo and put all your files on your CAE network file space. You must have a CAE account and your design files at your demo session to get credit!

Design Entry

For this course, we enter designs by drawing a schematic diagram that consists of a set of parts that are wired together ("schematic capture"). Altera provides a parts library for schematics, including gates and FPGA-specific parts such as input and output buffers and pads. For this class, you may also be given hardware description language (HDL) files to use for some homeworks. Although HDLs are now the primary means of specifying digital designs, it is important to first understand how hardware works, which is what you learn in this course. ECE 551 teaches hardware design using HDLs (352 is a prerequisite for it). If you are given HDL files to use, you will also receive step by step instructions on how to use them.

Synthesis

After we enter our design, the Quartus tool converts it into a netlist: a textual description of the components in the circuit and their interconnections. The process of synthesis converts both schematics and HDL files into the netlist. We will not cover the synthesis process, as Quartus does it automatically. The ECE/CS 556 course discusses this type of process.

Functional Simulation

Once the netlist is created, it can be simulated to determine if it functions correctly. This simulation does not deal with timing since circuit delay estimates are not available at this point in the design process. The tool allows graphical entry of simulation inputs and graphical observation of simulation outputs. At this point, we can check that the circuit generates the expected outputs for the given circuit inputs.

Implementation

Once we have a netlist and have verified its function, we can then perform implementation, which produces a file that can be loaded into an Altera FPGA to prototype the design (ie, actually implement the hardware circuit). The new file describes our circuit in terms of the FPGA's internal logic elements. Implementation consists of five main steps:

Map: The tool optimizes our circuit (eliminating unnecessary logic), and then breaks the circuit into pieces that can fit into a logic element (LE)

Place: The pieces from the mapping step are assigned to specific LEs and other FPGA resources. Placement tries to simplify the next step (routing) by assigning pieces with many interconnections to LEs that are near each other.

Route: All interconnections between the pieces of the circuit and the inputs/outputs are now assigned to specific routing resources of the FPGA. In other words, we connect the pieces that were placed in the previous step.

Timing: After mapping and place and route, locations of the look-up tables, flip-flops, input, outputs and all of the interconnection paths are known. Based on this (particularly the number of transistors along each interconnection path), delay estimates for various paths through the circuit and delay statistics for the FPGA can be calculated. These values are placed in a file which is used to perform timing simulation.

Configuration: The final implementation step is the generation of the configuration file—the sets of 1s and 0s to load into the FPGA configuration SRAM that implements the circuit. The file we use is an SOF (*SRAM object file*). When this file is loaded into the FPGA's configuration SRAM, the look-up tables are filled, the multiplexers are set, and the routing transistors turned ON or OFF to create the interconnections.

All of these steps produce report files that record what happened during that step, related statistics, and whatever warning or error messages were output (if any). A report viewer is provided to look at the reports. Another file produced by place and route is the Pin file that indicates which pin is assigned to each of the FPGA inputs and outputs. This file is useful for making board-level connections to the FPGA chip and knowing which pins were used for what purpose.

Timing Verification

After implementation, the delay estimates are available to permit timing simulation of the FPGA. Timing simulation permits detection of errors associated with the length of the delay paths in the circuit and can be used to determine whether or not performance specifications (speed requirements) are met. In addition, since it is performed after implementation, it also can be used to check for functional errors introduced in the implementation process.

Prototyping

Prototyping is the final verification step. With actual hardware available, it is possible to apply large numbers of inputs rapidly, far faster than can be done in simulation. Also, delay can be accurately measured, and the clock frequency can be varied for more detailed tests. For simple combinational and sequential circuits, we can apply enough combinations or sequences to fully demonstrate the function is correct. But for more complex circuits, this is impossible. Further, due to the limited time we have available, testing of such complex circuits must be selective.

At any one of the major steps listed, you may discover a problem with the design. To correct the problem, you will usually have to return to the design entry step, modify the design, and repeat most of the above steps. When this occurs at the prototyping stage, the number of steps to be repeated is high, so it is important to find problems as early as possible using very thorough simulation at the functional level to minimize the length of the entire design cycle.

2.2 Prototyping Hardware

We will use an Altera Cyclone IV FPGA board for our hardware demonstrations. The boards are powered by an external power supply (and two on-board voltage regulators), and include many resources attached to the FPGA for use in a wide variety of projects. In this homework, we will use only a limited subset of these resources. Notably, we will use small slide switches, LEDs, seven-segment LED displays, and pushbuttons on the board that operate through an IC that eliminates contact bounce problems.

We will use the pin assignment function in Quartus to ensure that the inputs and outputs of our circuit are correctly mapped to FPGA pins that connect to the intended devices on the board. It is important to verify that you have named your external connections correctly to prevent damage to the FPGA or the external devices. Please be **very careful** in using these boards, follow instructions carefully, and verify everything **before** powering up the board.

Further information on the boards and Quartus is available on the website, including the DE2-115 User Manual, the 352 Board Diagram image (which shows all of the connections you need and the locations of the pushbuttons, switches, and most of the LEDs), and the Quartus FAQ, which provides helpful hints and answers to common questions about using Quartus.

Altera Cyclone IV Series FPGAs

The information in this box is not needed to complete this homework. It is provided in case you are interested.

Look-up tables (LUTs) implement combinational logic functions in most current FPGA architectures. A LUT is a very small static random-access memory (SRAM) that holds the truth table for the function it implements. The inputs of the LUT act as an address for the SRAM, and the output is the value held at that address (that line of the truth table). A complete logic circuit is formed by programming the contents of the LUTs as needed, then setting other SRAM bits that control how the LUTs are connected to one another through tristates and multiplexers. These values are loaded during FPGA configuration (generally each time the FPGA is powered on), which can take 100s of milliseconds, depending on the size of the FPGA.

The LUTs in the Cyclone FPGA are 4-LUTs: they have four inputs. Any logic function with 4 inputs and 1 output can be implemented using a single 4-LUT. The fundamental structure of the Cyclone FPGA is known as a logic element (LE), shown below in Figure 1, and each LE contains 4-input LUT and a flip-flop (a storage element we'll discuss later).

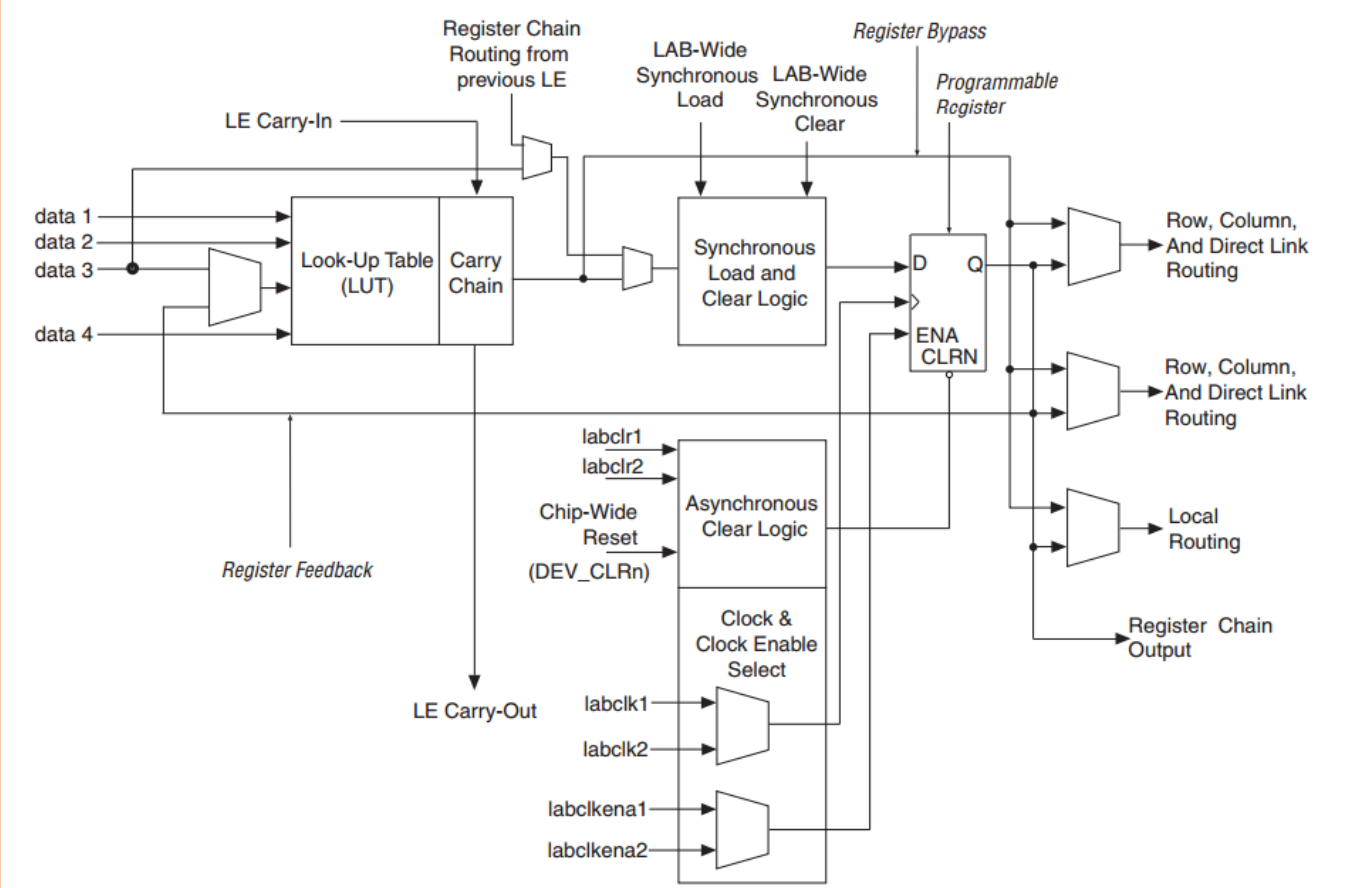

Figure 1: Altera Cyclone IV Logic Element (LE) [1]

The process of deciding which LUT in the FPGA should implement which logic gate, and which wires should implement which signals between gates is very complex. These choices affect whether or not the circuit fits in the FPGA, and if it does, how fast it can run. Fortunately, this is done automatically for us by the Quartus development tool. In a realistic commercial design, you would probably have to "tweak" certain tool parameters to meet your particular design requirements and produce an "optimal" design.

[Remember "optimal" will mean different things for different designs and situations (minimal delay, minimal area, minimal power, minimal skew, etc). This is why companies hire highly-paid hardware designers!]

3 Applied Homework Tasks

This must be done ***individually***. Do not just watch someone else do the tutorial. You need to do the tutorial yourself to help you understand how to use the tool, which you will need to use for written and applied homework throughout the semester. All work submitted for AHW1 **must be your own work**, based on working through the tutorial and this document.

Be sure to read the earlier part of this document so that you have an idea of the steps required for design of circuits using FPGAs and what you are doing in the tutorial.

In this experiment, you will design and test an FPGA-based "full adder" – a circuit that adds two bits and produces a sum bit and a carry bit, and use it to create a larger adder. Later during the demo, we will configure this circuit onto the FPGA board and use LEDs to display the input and output values.

3.1 Accessing Quartus

You can access Quartus on any CAE machine or you can install it on your own (but make sure you are running 16.0

- If you access Quartus from a CAE machine, search in the programs menu for Quartus.
- If you install it on your own machine, note that it is a large application.

3.2 Quartus Tutorial

1) Download the AHW1.zip file from the course website. Extract the contents of the zip archive to the following location on your CAE network storage: I:\ECE352\AHW\AHW1
2) Open the Quartus project double clicking the Quartus project file in the directory above in Windows explorer
3) For this tutorial, you will create a full-adder by placing logic gates in a schematic sheet. You can create a new schematic sheet in Quartus by clicking on the New File icon in the task bar *or* from File→New. A dialog will open and you should select "Block Diagram/Schematic File".

   Creating a New File

4) Schematics are normally viewed in landscape mode, so you will configure the new schematic file in land scape mode by navigating to File→Page Setup. In the dialog that appears, select Landscape and click OK.

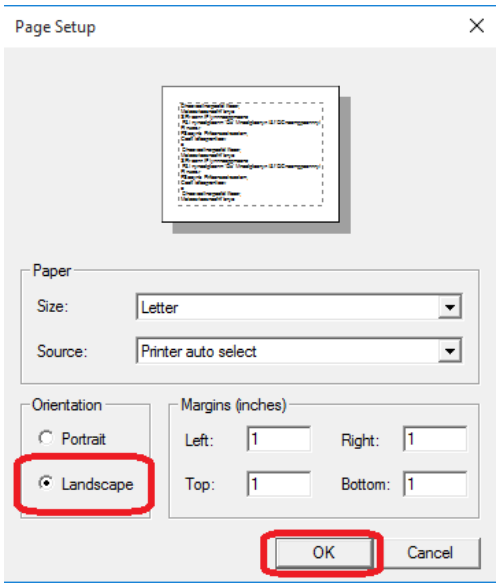

Figure 2

5) Save the file as fa.bdf. This can be done by navigating to File→Save As.

6) In the Project Navigator pane, change the pull down menu that is currently set as "Hierarchy" so that it says "Files".

7) Before you add anything to fa.bdf, you will want to tell Quartus that fa.bdf is the top level design. This can be done by selecting "Files" in the project navigator pane, then by right clicking on fa.bdf, and selecting "Set as Top Level Entity".

8) Now that the page is set up, you will need to place the gates required to implement a full adder. A full adder requires two 2-input AND gates, two 2-input XOR gates, and one 2-input OR gate. Gates can be placed in a schematic by clicking on the "Symbol Tool" or by double clicking the left mouse button in the schematic.

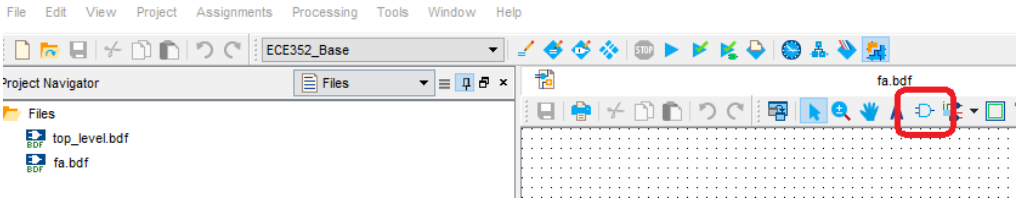

Figure 3

From the dialog, expand the following drop downs altera_lite→primatives→logic.

In this pull-down there are a list of gates that you can select from to add to you schematic. Add two(2) instances of ***and2***, two (2) instances of ***xor***, and a single (1) instance of ***or2***. You can press the escape key (ESC) at any time to stop placing gates.

9) Now that the gates are placed, you will need to place three (3) ***input*** pins to the left of your schematic and two (2) ***output*** pins to the right. Input/output pins are found under altera_lite→primatives→pin. When you have completed this step, your schematic should look like Figure 4.

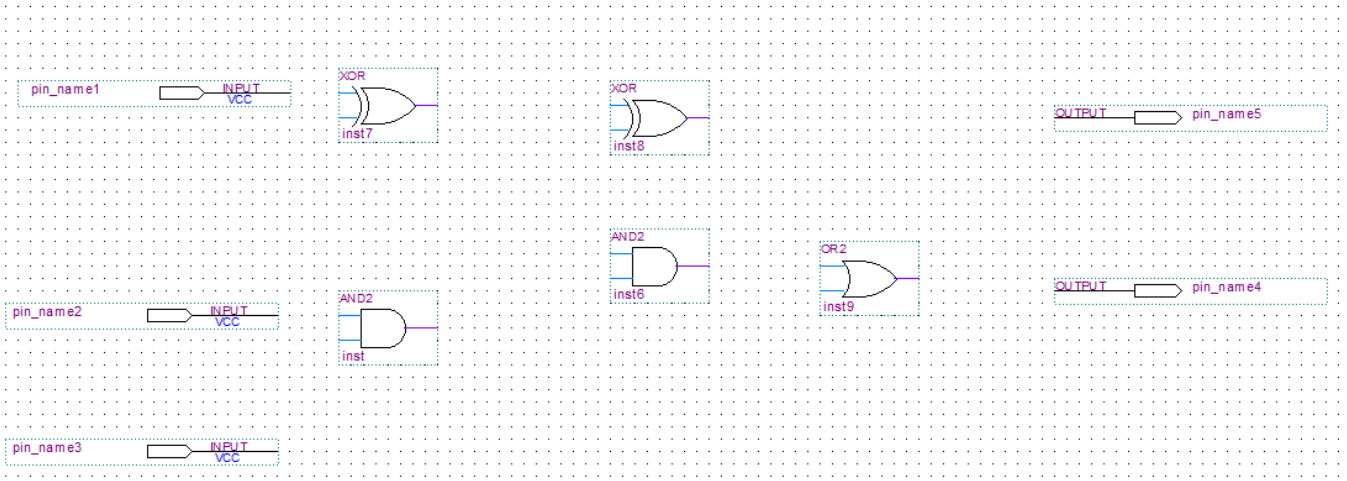

Figure 4

The video link below shows how to add the gates in the previous image.

Placing Gates

10) Now you will assign names to the input and output pins. You can change the name of a pin by double clicking on the pin and providing a name. Use the names listed below to name your input/output pins.

Input Pin Names (Top to Bottom)	Output Pin Names (Top to Bottom)
A	**S**
B	**Cout**
Cin	

11) Now that the gates and inputs are placed, you need to interconnect the gates to create the full adder. You can place wires by selecting the "Orthogonal Node Tool" or by hovering over an input pin with the cursor.

Figure 5

Wire your circuit so that it looks like Figure 6.

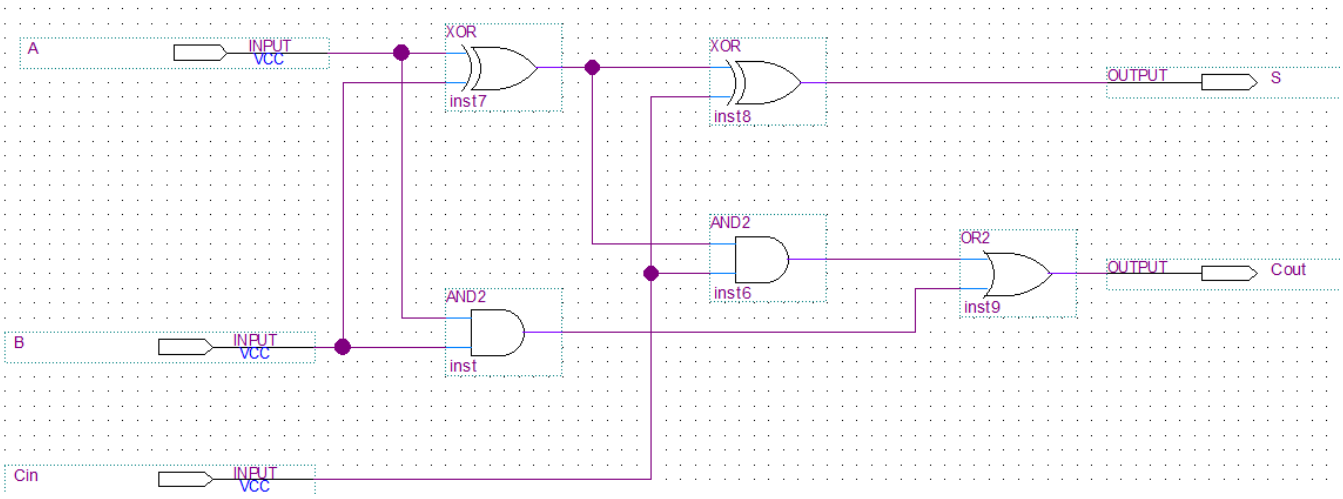

Figure 6

The following video demonstrates how to connect your gates using wires.

Wiring Gates

12) When you have finished, be sure to save your circuit (File→Save).

13) Now that your circuit is completed, you need to compile the circuit. This can be done using the "Start Compilation" button. Be sure to examine the *Messages* output pane and correct any errors that are reported.

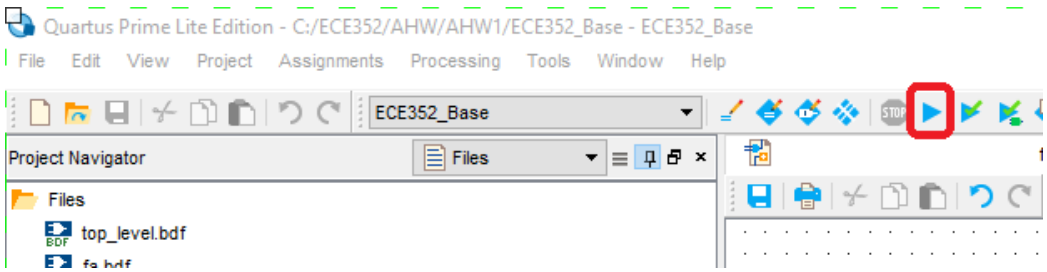

Figure 7

14) Now that your full adder has been compiled, you will want to simulate the circuit and verify the results. Quartus provides a waveform editor that will allow you to specify the inputs to the circuit and Quartus will generate the corresponding outputs. You will create a new waveform file by selecting File→New.

15) From the dialog, select Verification/Debugging Files→University Program VWF.

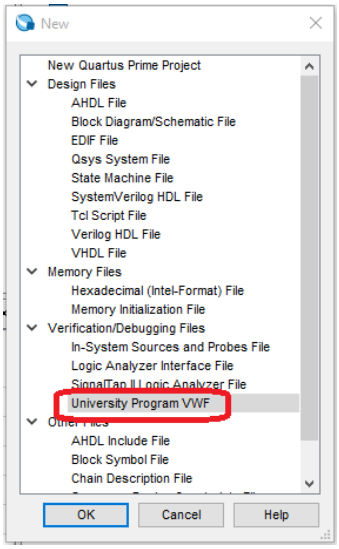

Figure 8

16) In the new window that appears, you will need to add the inputs and outputs for your circuit. This can be by navigating to Edit→Insert→Insert Bus or Node.

17) From the new dialog, select "Node Finder"

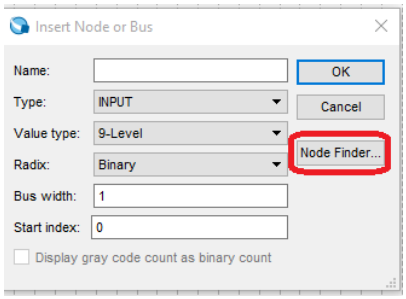

Figure 9

18) Ensure that the Filter Input is set to “Pins:all” and then select the “List” button.

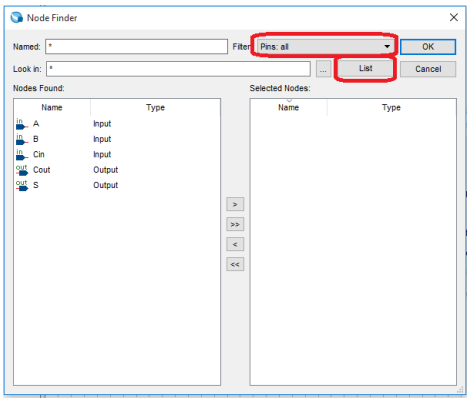

Figure 10

19) You will want to view all inputs and output for this circuit, so click the Select All (>>) button. Once all the pin names are listed in the “Selected Nodes” entry box, click OK. Click OK on the dialog box that is still open.

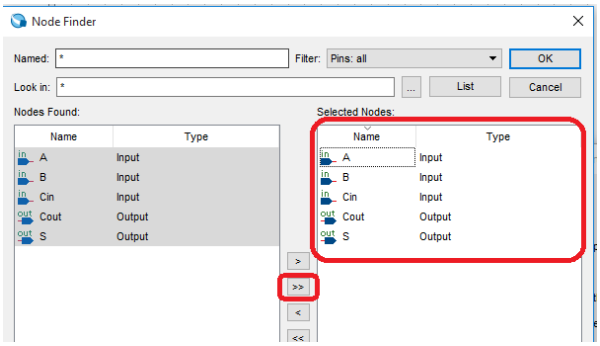

Figure 11

20) In order to exhaustively test all 8 possible input combinations for our 3 input pins, you will use “Overwrite Clock” icon to set the values on the A, B, and Cin inputs. The only field in the Clock dialog that you should change is the period. The period of the three inputs should be set as follows.

Input	Period
A	240ns
B	480ns
Cin	960ns

21) Save the file as fa.vwf.

22) You can simulate the file by selecting Simulation→Run Functional Simulation. Verify that your functional simulation matches the behavior of a full adder. If it does not match the expected behavior, examine your circuit any errors.

This video shows you the process of compiling and simulating the full adder.

Simulating a Circuit

23) Save an image of the simulation using Windows Snipping Tool. Save the file as **`fa_func_sim.jpg`.**

24) Open your fa.bdf and add a title block to your schematic. **Every schematic, in every applied homework, is required to contain a title block**. Title blocks can be placed in a similar fashion as a gate. A title block is found in altera_lite→primitives→other. Fill in your name for the DESIGNER and the name of the circuit for the TITLE.

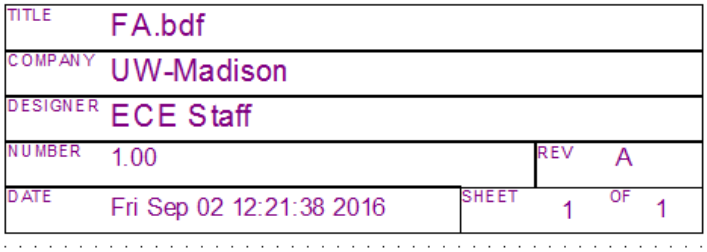

Figure 12

25) Save your fa.bdf file.

26) Now you will create a symbol for your 1-bit full adder. You can create a file by selecting File→Create/Update→Create Symbol Files for Current File. Save the symbol as fa.bsf.

27) Create a new schematic file (File→New→Block Diagram/Schematic File).

28) Set the file properties to be in landscape mode (File→Page Setup).

29) Save the file as fa4.bdf.

30) Now you will add 4 instances of your 1-bit full adder to fa4.bdf. Use the "Symbol" icon to place the full adder. The full adder will be found above the entries for altera_lite.

When you're finished added fa symbols, your fa4.bdf file should look like Figure 13.

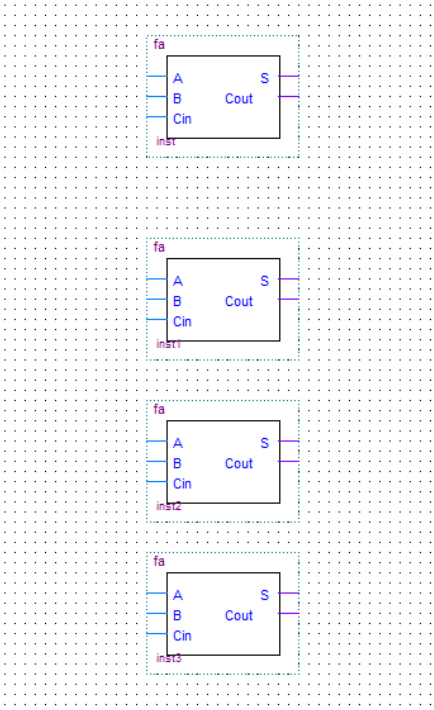

Figure 13

The video below demonstrates how to generate the fa.bsf symbol and create fa4.bsf.

Adding a Full Adder to a Schematic

31) Now that the fa blocks are placed, you will need to place three (3) ***input*** pins to the left of your schematic and two (2) ***output*** pins to the right.

32) Now you will assign names to the input and output pins. Since this is a 4-bit adder, the inputs A, B, and S all need to be 4-bits wide. A multi-bit wide connection is called a bus. A bus is designed by brackets [] and the width of the bus is determined by the starting and ending numbers. Be sure in include the two periods ".." in between the starting and ending widths of the bus! Use the names listed below to name your input/output pins.

Input Pin Names (Top to Bottom)	Output Pin Names (Top to Bottom)
A[3..0]	**S[3..0]**
B[3..0]	**Cout**
Cin	

The following video shows you how to add the input pins and then name inputs A, B, and S as 4-bit wide busses.

Multi-Bit Inputs (Busses)

33) Now you will connect the busses for the A, B, and S signals of your full adders. You can add a bus by selecting the Orthogonal Bus Tool Icon or by simply hovering over and input/output that has already been defined as a bus. Follow along with the next video to see how to add these three busses.

Connecting Busses

34) You now have to route the individual wires from the 4-bit bus to the corresponding inputs of the full adders. In order to properly identify each bus, single click on each bus and give it a name that matches the name provided to the input pin.

35) Connect a wire from the A input of each full adder to the A[3..0] bus.

36) Connect a wire from the B input of each full adder to the B[3..0] bus.

37) Connect a wire from the S output of each full adder to the S[3..0] bus.

38) On the top most full adder, single click on the wire connecting the A input to the A[3..0] bus. The bus should turn blue. Using the keyboard, type A[0]. This has named the wire so that the least significant bit of the A[3..0] bus is connected to the A input of the top most full adder.

39) Name the remaining wires connected to the A[3..0], B[3..0], and S[3..0] to match Figure 14.

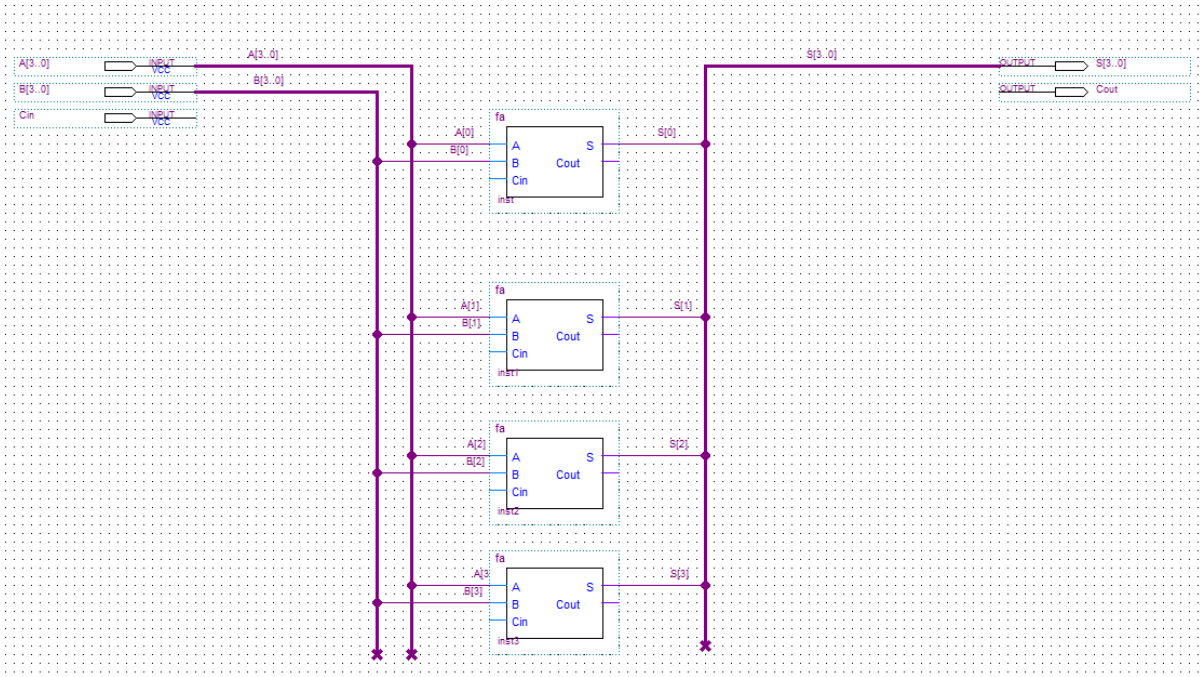

Figure 14

40) Connect the Cin input pin to the Cin of the least significant bit of the 4-bit full adder.

41) You will also need to connect the carry out from each bit of the 4-bit full adder to the Cin of the next full adder.

42) The Cout of the most significant bit should be connect to the Cout output pin.

43) Add a title block to your schematic. Your schematic should look like Figure 15.

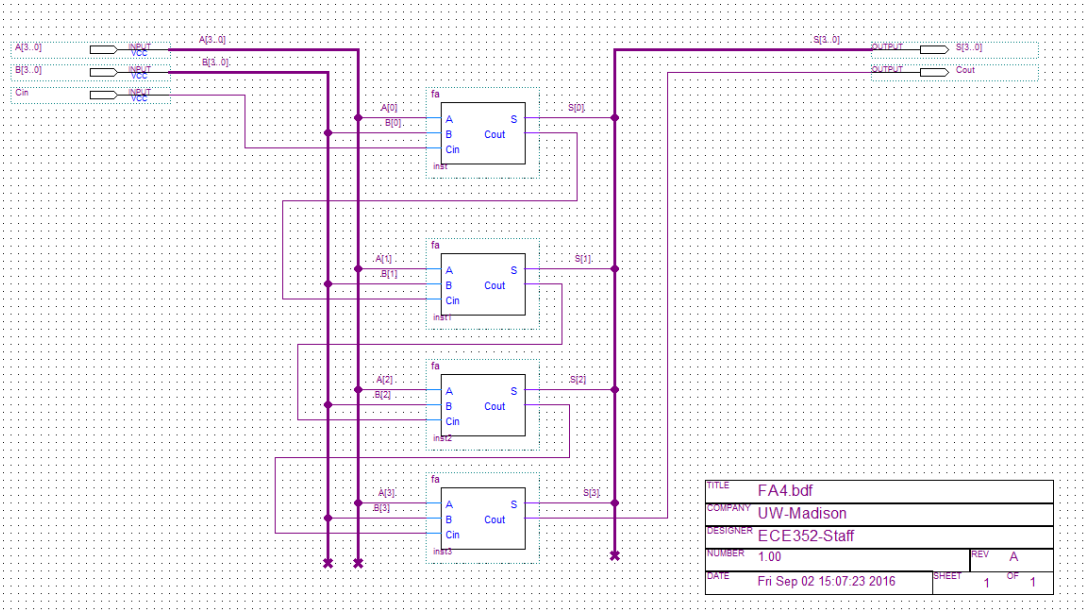

Figure 15

You can follow these steps in the video below

[Naming Bus Nets]

44) In the project navigator, right click on fa4.bdf and set it as the Top Level Entity.

45) Compile the design and verify that there were no errors.

46) Now you will need to simulate the 4-bit adder. You will create a new waveform file by selecting File→New. From the dialog, select Verification/Debugging Files→University Program VWF.

47) Add all the Inputs/Outputs to the simulation using the Node Finder.

48) Set the radix of A[3..0], B[3..0] and S[3..0] to be hexadecimal.

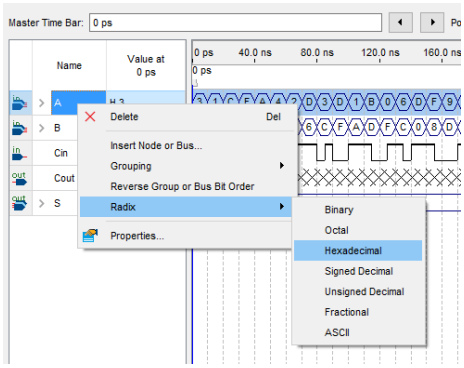

Figure 16

49) You will now randomly assign values to the inputs A[3..0], B[3..0], and S[3..0] by clicking on each input pin and then on the Random Value icon as shown below. For the A and B inputs, select the radio box for "Every grid interval". For Cin, select "Every half grid interval"

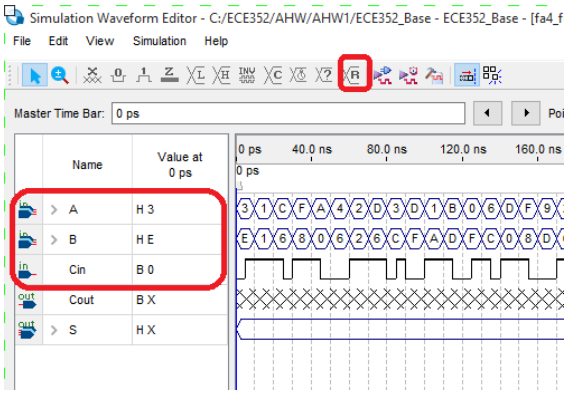

Figure 17

50) Save the file as fa4_func_sim.vwf.

51) Functionally simulate the circuit and verify that the 4-bit adder functions correctly. If your circuit does not operate correctly, review you design and fix any errors that you find.

Simulate 4-bit Full Adder

52) Save an image of the simulation using Windows Snipping Tool. Save the file as **`fa4_func_sim.jpg`**. Annotate the simulation results for the first four values applied to A[3..0], B[3..0], and C. The annotations should list the decimal values being added together and also if a carryout was generated.

53) Now that your 4-bit adder is functionally correct, we need to connect it to some of the switches and LEDs on the DE2-115 development board that will be used in lab. In order to do this, generate a symbol for the four bit adder called fa4.bds.

54) In the project navigator, right click on topl_level.bdf and set it as the Top Level Entity.

55) Open top_level.bdf and add a single instance of your 4 bit full adder.

56) The default ECE352 project has predefined names for the pins connected to the push buttons, switches, and LEDs that are connected to the FPGA on the DE2-115 development board. You can view the names and pin assignments by selecting Assignments→Pin Planner. InFigure 18, a filter of "SW*" has been entered to show only the pins connected to the 18 slide switches. **Close the Pin Planner without making any modifications.**

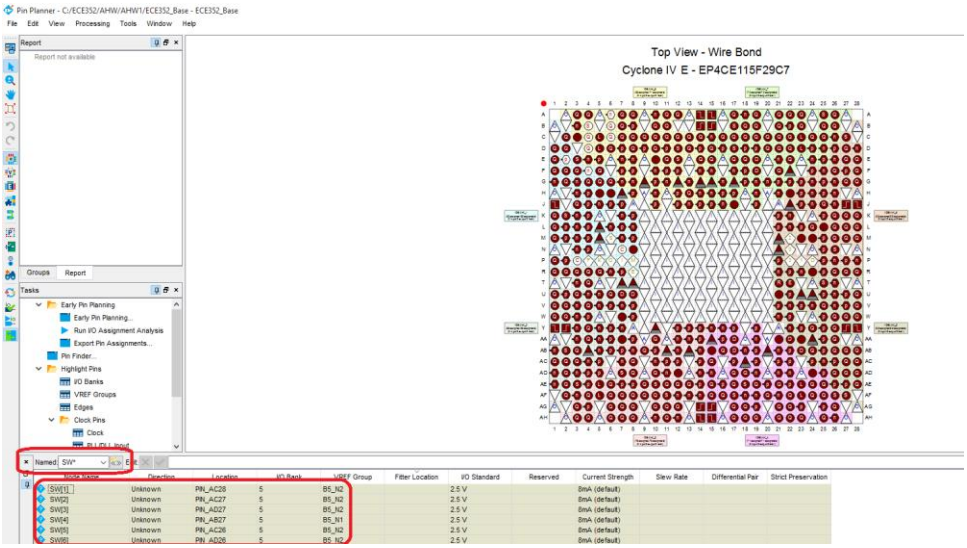

Figure 18

57) Add one input pin and name it SW[17..0]. SW[17..0] is the name that has been given to the pins that are connected to the 18 slide switches on the DE2-115.

58) Add two output pins. Name them LEDR[17..0] and LEDG[7..0]. These are the names given to the pins connected to the 18 red LEDs and 8 green LEDs on the DE2-115.

59) Add a single instance of a 'wire' buffer (altera_lite→primatives→buffer→wire or just search for 'wire' in Symbol dialog search field). The wire buffer simply passes the signal from the input to the output.

60) Complete top_level.bdf so that it looks like Figure 19. This schematic uses connection by name when connecting the full adder to the input pins. When two wires have the same name, Quartus assumes that the two wires are connected. Connection by name allows for cleaner, more readable schematics.

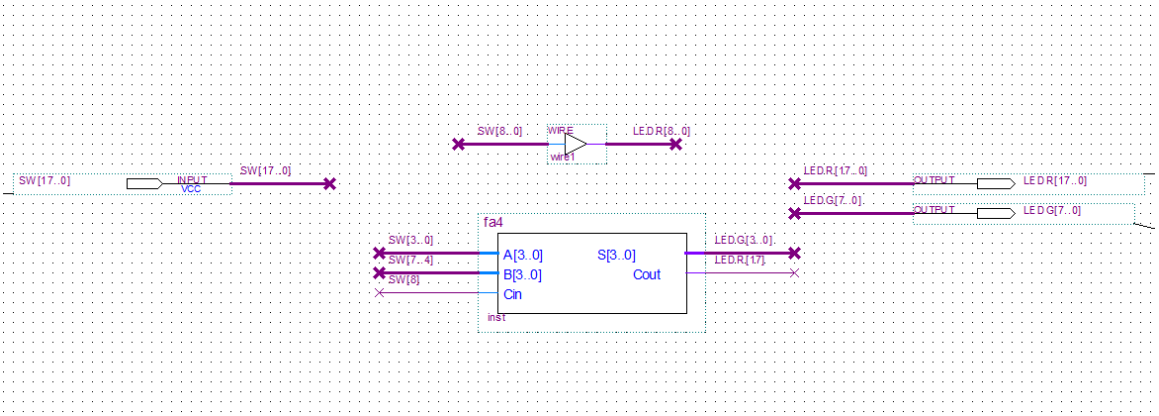

Figure 19

61) In your lab demo, you will need to compile and load your project onto the DE2-115. The image below shows you how to load a project onto the DE2-115. Make sure that the USB-Blaster is listed in the Hardware Setup prior to programming. The video will show you how to select the USB-Blaster if it is not listed.

62) Verify that the .sof file is pointing to the current directory. There shouldn't be any characters preceding ECE352_Base.sof.

63) If the USB-Blaster is listed, select the Start button. As the FPGA programs, you will see the progress indicator go from 0% to 100%.

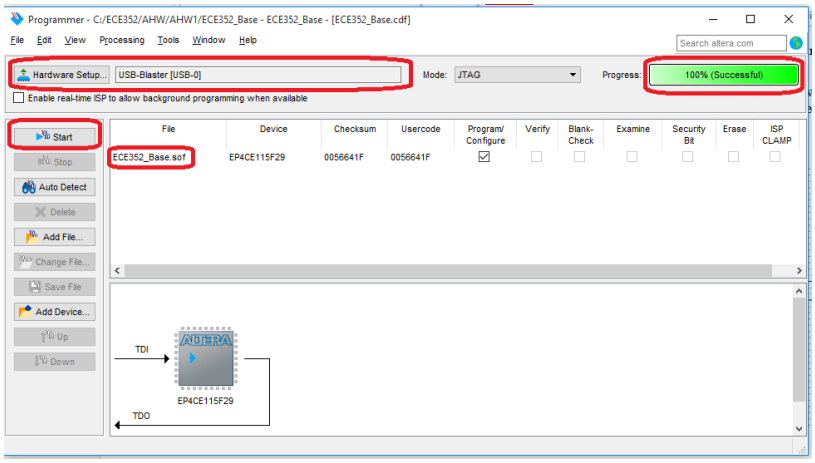

Figure 20

[Programming the FPGA]

64) **Assemble the files listed at the end of this document, and upload them to Moodle in the AHW1 activity.** You will need to scan your annotated waveforms so that you can upload them. **You must submit all files by the Applied Homework deadline listed on the class webpage.** Later, in your demo session (lab), you will further verify and demonstrate your circuits by loading them onto an FPGA board so that you can use the switches to control the circuit inputs, and the LEDs to see the outputs of the circuit.

Submission Notes:

- **For any schematics you turn in this semester for any written or applied homework**, **you must include a title block**. The title block **must** include the name of the circuit and your name. For later AHWs where you can work with a partner, you will list both partner names here.
- **For any waveforms you turn in this semester**, ensure that your waveforms are zoomed in enough that they can be read, and should include the requested span of time and any requested annotations. If no specific timespan is given for a combinational circuit's waveform, show each tested input combination **once**. Scan the waveform printout and save it as a JPG or PDF. The resolution of submitted JPGs should be no more than 1920x1080.

4 Applied Homework Submission

You will upload a number of files to the AHW1 activity in Moodle. You will also need to answer a number of questions in that activity. Follow the directions in the activity carefully, and be sure that your files are named correctly.

You will submit the BDF files (which should each include a title block) from the tutorial. You will also submit a JPG image of the BDF. Each JPG should have the same base file name as the associated BDF (e.g., the JPG image of **`fa.bdf`** should be named **`fa.jpg`**). To create a JPG for a BDF, in Quartus go to File → Export. The required files include:

- **`fa.bdf`** and **`fa.jpg`** – the full adder schematic and its JPG
- **`fa4.bdf`** and **`fa4.jpg`** – the 4-bit ripple-carry adder schematic and its JPG
- **`top_level.bdf`** and **`top_level.jpg`** – the top level schematic and its JPG

You will also upload **scanned** copies of the waveforms that you have printed and annotated according to the instructions in this tutorial document. Scans of waveforms can be submitted as either PDF or JPG files. Do not use any other file format. Be sure that the scans are **legible**, and limit JPGs to no more than 1920x1080 resolution. The required waveform scans include:

- **`fa_func_sim.jpg`** (or **`fa_func_sim.pdf`**) – the functional simulation waveform for the full adder circuit
- **`fa4_func_sim.jpg`** (or **`fa_func_sim.pdf`**) – the functional simulation waveform for the 4-bit full adder circuit

This submission is due by the Applied Homework deadline listed on the class webpage.

Note that this deadline is in the week BEFORE the related demo session!

5 Applied Homework Demonstration

The following step in the tutorial will be completed in lab.

1. **After** you have successfully compiled your 4-bit full adder, you will load the resulting image onto the DE2-115 development platform. Use switches SW[8..0] to supply inputs to your 4-bit full adder. Observe the resulting 4-bit number on LEDG[3..0]. Demonstrate to you TA that your 4-bit full adder functions correctly.
2. After your TA has verified your 4-bit full adder, you will be given a 3-input Boolean equation to implement in Quartus.

   The following equation is an example of the type of equation that will be given to you: $F = AB + A\overline{C} + \overline{A}C$
3. After you have completed your circuit for the Boolean equation, simulate the circuit and exhaustively verify all possible input combinations. Your waveform should supply the inputs in the following order

	Inputs		
Order	A	B	C
1	0	0	0
2	0	0	1
3	0	1	0
4	0	1	1
5	1	0	0
6	1	0	1
7	1	1	0
8	1	1	1

4.

References

[1] Cyclone IV Device Handbook,, Altera Corporation, Volume 1, Chapter 2, pp. 30, May 2009.